12

Simple Filters

This chapter features the simple filters of the system—commands which accept data from standard input, manipulate it and write the results to standard output. Filters are the central tools of the UNIX tool kit, and each filter featured in this chapter performs a simple function. This chapter shows their use both in standalone mode and in combination with other tools using redirection and piping.

Many UNIX files have lines containing *fields*—strings of characters representing a meaningful entity. Some commands expect these fields to be separated by a suitable delimiter that's not used by the data. Typically this delimiter is a : (as in `/etc/passwd` and `$PATH`), but we have used the | (pipe) as delimiter for some of the sample files in this and other chapters. Many filters work well with delimited fields, and some simply won't work without them.

What You Will Learn

- Use **pr** to format text to provide margins and headers, doublespacing and multiple column output.
- Pick up lines from the beginning with **head**, and from the end with **tail**.
- Extract characters or fields with **cut**.
- Join two files laterally, and multiple lines to a single line with **paste**.
- Sort, merge and remove repeated lines with **sort**.
- Find out the unique and nonunique lines with **uniq**.
- Change, delete or squeeze individual characters with **tr**.

Topics of Special Interest

- A special feature of **paste** to form a singe line from multiple lines.
- The counting facility available in **uniq**.
- Use all of these commands in an example to perform content manipulating tasks.

Filters Reviewed

Filters were introduced in Section 8.5.4 as a category of commands that take advantage of the shell redirection features. A filter has the capability of reading standard input and writing to standard output. By default, a filter writes to standard output. It reads from standard input when used without a filename as argument, and from the file otherwise.

The piping mechanism of the shell lets the standard output of one filter serve as standard input to another. This feature lets us design pipelines containing a series of filters. Section 12.10 shows these tools in combination for performing content manipulations tasks—tasks which these tools can't do when acting alone.

12.1 THE SAMPLE DATABASE

Henceforth, you'll be learning the features of several UNIX commands, including the advanced ones, text editing and shell programming with reference to a file `emp.lst`. It's a good idea to have a close look at the file now and understand the organization:

```
$ cat emp.lst
2233|a.k. shukla          |g.m.      |sales      |12/12/52|6000
9876|jai sharma           |director  |production |12/03/50|7000
5678|sumit chakrobarty    |d.g.m.    |marketing  |19/04/43|6000
2365|barun sengupta       |director  |personnel  |11/05/47|7800
5423|n.k. gupta           |chairman  |admin      |30/08/56|5400
1006|chanchal singhvi     |director  |sales      |03/09/38|6700
6213|karuna ganguly       |g.m.      |accounts   |05/06/62|6300
1265|s.n. dasgupta        |manager   |sales      |12/09/63|5600
4290|jayant Choudhury     |executive |production |07/09/50|6000
2476|anil aggarwal        |manager   |sales      |01/05/59|5000
6521|lalit chowdury       |director  |marketing  |26/09/45|8200
3212|shyam saksena        |d.g.m.    |accounts   |12/12/55|6000
3564|sudhir Agarwal       |executive |personnel  |06/07/47|7500
2345|j.b. saxena          |g.m.      |marketing  |12/03/45|8000
0110|v.k. agrawal         |g.m.      |marketing  |31/12/40|9000
```

This is a text file designed in fixed format and containing a personnel database. There are 15 lines in the file, where each line has six fields separated from one another by the delimiter |. The details of an employee are stored in one line. A person is identified by the emp-id, name, designation, department, date of birth and salary, as indicated by the fields (in the same order). You'll be using this files, or ones derived from them, in various ways to see the extent of manipulation that is possible with the UNIX tool kit.

12.2 pr: PAGINATING FILES

The pr command prepares a file for printing by adding suitable headers, footers and formatted text. A simple invocation of the command is to use it with a filename as argument:

```
$ pr dept.lst

May 06 10:38 1997  dept.lst Page 1

01:accounts:6213                    These six lines are the original contents of
02:admin:5423                       dept.lst shown in Section 5.1
03:marketing:6521
04:personnel:2365
05:production:9876
06:sales:1006

...blank lines...
```

pr adds five lines of margin at the top and five at the bottom. The lower portion of the page has not been shown in the examples for reasons of economy. The header shows the date and time of last modification of the file along with the filename and page number.

12.2.1 pr Options

pr's *-k* option (where *k* is an integer) prints in *k* columns. If a program outputs a series of 20 numbers, one in each line, then this option can make good use of the screen's empty spaces. And because pr is a filter, it can obtain its input from the standard output of another program. Let's use the -t option also to suppress the headers and footers:

```
$ a.out | pr -t -5
0       4       8       12      16
1       5       9       13      17
2       6       10      14      18
3       7       11      15      19
```

If you are not using the -t option, then you can have a header of your choice with the -h option. This option is followed by the header string. There are some more options that programmers will find useful:

- -d Doublespaces input, reduces clutter.
- -n Numbers lines, helps in debugging code.
- -o *n* Offsets lines by *n* spaces, increases left margin of page.

Combine these various options to produce just the format you need:

```
$ pr -t -n -d -o 10 dept.lst
          1    01:accounts:6213

          2    02:admin:5423
```

```
3    03:marketing:6521
4    04:personnel:2365
5    05:production:9876
6    06:sales:1006
```

There's one option that uses a number prefixed by a + to print from a specific page number. Another option (-l) sets the page length:

```
pr +10 chap01                    Starts printing from page 10
pr -l 54 chap01                  Page length set to 54 lines
```

Because **pr** formats its input by adding margins and a header, it's often used as a "pre-processor" before printing with the **lp** command:

```
pr -h "Department list" dept.lst | lp        Use lpr in Linux
```

Since **pr** output often lands up in the hard copy, **pr** and **lp** form a common pipeline sequence.

Note: For numbering lines, you can also use the **nl** command (not covered in this edition). It's easier to use **nl foo** than **pr -t -n foo**.

12.3 head: DISPLAYING THE BEGINNING OF A FILE

The **head** command, as the name implies, displays the top of the file. When used without an option, it displays the first ten lines of the file:

```
head emp.lst                                 Shows first ten lines
```

You can use the -n option (POSIX mandated) to specify a line count and display, say, the first three lines of the file:

```
$ head -n 3 emp.lst                          Or head -3 emp.lst on some systems
2233|a.k. shukla       |g.m.      |sales      |12/12/52|6000
9876|jai sharma        |director  |production |12/03/50|7000
5678|sumit chakrobarty|d.g.m.    |marketing  |19/04/43|6000
```

head can be used in imaginative ways. Consider that you are resuming an editing session the next day and find that you are unable to recall the name of the file you last edited. Since **ls -t** displays filenames in order of their modification time, the job is easily done by picking up the first filename from the list and using it as an argument to **vi**. This requires command substitution:

```
vi `ls -t | head -n 1`                       Opens last modified file for editing
```

You can define this as an *alias (10.4)* so the aliased command is always available for you to use

12.4 tail: DISPLAYING THE END OF A FILE

Complementing its **head** counterpart, the **tail** command displays the end of the file. It provides an additional method of addressing lines, and like **head**, it displays the last ten lines by default. The last three lines are displayed in this way:

```
$ tail -n 3 emp.lst                                    Or use tail -3 emp.lst
3564|sudhir Agarwal   |executive|personnel |06/07/47|7500
2345|j.b. saxena      |g.m.     |marketing |12/03/45|8000
0110|v.k. agrawal     |g.m.     |marketing |31/12/40|9000
```

You can also address lines from the beginning of the file instead of the end. The *+count* option allows you to do that, where *count* represents the line number from where the selection should begin. Since the file contains 15 lines, selecting the last five implies using

```
tail +11 emp.lst                          11th line onwards, possible with + symbol
```

12.4.1 tail Options

tail has more options than **head**, and can also extract in units of bytes rather than lines.

Monitoring File Growth (-f) Many UNIX programs constantly write to the system's log files as long as they are running. System administrators need to monitor the growth of these files to view the latest messages. **tail** offers the **-f** (follow) option for this purpose. This is how you can monitor the installation of Oracle by watching the growth of the log file **install.log** from another terminal:

```
tail -f /oracle/app/oracle/product/8.1/orainst/install.log
```

The prompt doesn't return even after the work is over. With this option, you have to use the interrupt key to abort the process and exit to the shell.

Extracting Bytes Rather than Lines (-c) POSIX requires **tail** to support the **-c** option followed by a positive or negative integer depending on whether the extraction is performed relative to the beginning or end of a file. Solaris supports this option only in its XPG4 version, but this is no problem in Linux:

```
tail -c -512 foo          Copies last 512 bytes from foo
tail -c +512 foo          Copies everything after skipping 511 bytes
```

Tip: Use **tail -f** when you are running a program that continuously writes to a file, and you want to see how the file is growing. You have to terminate this command with the interrupt key.

12.5 cut: SLITTING A FILE VERTICALLY

The features of the **cut** and **paste** commands will be illustrated with specific reference to the file **shortlist**, which stores the first five lines of **emp.lst**:

```
$ head -n 5 emp.lst | tee shortlist
2233|a.k. shukla      |g.m.     |sales      |12/12/52|6000
9876|jai sharma       |director |production|03/12/50|7000
5678|sumit chakrobarty|d.g.m.   |marketing  |04/19/43|6000
2365|barun sengupta   |director |personnel  |05/11/47|7800
5423|n.k. gupta       |chairman |admin      |08/30/56|5400
```

Note the use of the **tee** facility that saves the output in `shortlist` and also displays it on the terminal. We can extract both columns and fields from this file with the **cut** command. Columns are specified with the `-c` option and fields with `-f`. We'll take up columns first.

12.5.1 Cutting Columns (-c)

To extract specific columns, you need to follow the `-c` option with a list of column numbers, delimited by a comma. Ranges can also be used using the hyphen. Here's how we extract the name and designation from `shortlist`:

```
$ cut -c 6-22,24-32 shortlist
a.k. shukla      g.m.
jai sharma       director
sumit chakrobartyd.g.m.
barun sengupta   director
n.k. gupta       chairman
```

Note that there should be no whitespace in the column list. Moreover, **cut** uses a special form for selecting a column from the beginning and up to the end of a line:

```
cut -c -3,6-22,28-34,55- shortlist          Must be an ascending list
```

The expression `55-` indicates column number 55 to end of line. Similarly, `-3` is the same as `1-3`.

12.5.2 Cutting Fields (-f)

The `-c` option is useful for fixed-length lines. Most UNIX files (like `/etc/passwd` and `/etc/group`) don't contain fixed-length lines. To extract useful data from these files you need to cut fields rather than columns. **cut** uses the tab as the default field delimiter, but can also work with a different delimiter. Two options need to be used here: `-d` for the field delimiter and `-f` for the field list. This is how you cut the second and third fields of our sample file:

```
$ cut -d \| -f 2,3 shortlist | tee cutlist1
a.k. shukla      |g.m.
jai sharma       |director
sumit chakrobarty|d.g.m.
barun sengupta   |director
n.k. gupta       |chairman
```

The `|` was escaped to prevent the shell from interpreting it as the pipeline character; alternatively it can also be quoted (`-d"|"`). To cut out fields numbered 1, 4, 5 and 6 and save the output in `cutlist2`, follow a similar procedure:

```
cut -d "|" -f 1,4- shortlist > cutlist2          Here | is quoted
```

Extracting User List from who *Output* **cut** can be used to extract the first word of a line by specifying the space as the delimiter. The example used in Section 3.10 now run in tandem with **cut** displays the list of users only:

```
$ who | cut -d " " -f1                    Space is the delimiter
root
kumar
sharma
project
sachin
```

cut is a powerful text manipulator often used in combination with other commands or filters. You'll be using the command a number of times in this text.

Note: You must indicate to **cut** whether you are extracting fields or columns. One of the options **-f** and **-c** must be specified. These options are really not optional; one of them is compulsory.

12.6 paste: PASTING FILES

What you cut with **cut** can be pasted back with the **paste** command—but vertically rather than horizontally. You can view two files side by side by pasting them. In the previous topic, **cut** was used to create the two files `cutlist1` and `cutlist2` containing two cut-out portions of the same file. Using **paste**, you can fix them laterally:

```
$ paste cutlist1 cutlist2
a.k. shukla       |g.m.      2233|sales      |12/12/52|6000
jai sharma        |director  9876|production |12/03/50|7000
sumit chakrobarty|d.g.m.     5678|marketing  |19/04/43|6000
barun sengupta    |director  2365|personnel  |11/05/47|7800
n.k. gupta        |chairman  5423|admin      |30/08/56|5400
```

The original contents have been restored to some extent, except that the fields have different relative locations, and pasting has taken place on whitespace. Like **cut**, **paste** also uses the tab as the default delimiter, but you can specify *one or more delimiters* with **-d**:

```
$ paste -d"|" cutlist1 cutlist2
a.k. shukla       |g.m.      |2233|sales      |12/12/52|6000
jai sharma        |director  |9876|production |12/03/50|7000
sumit chakrobarty|d.g.m.     |5678|marketing  |19/04/43|6000
barun sengupta    |director  |2365|personnel  |11/05/47|7800
n.k. gupta        |chairman  |5423|admin      |30/08/56|5400
```

Even though **paste** uses at least two files for concatenating lines, the data for one file can be supplied through the standard input. If, for instance, `cutlist2` doesn't exist, you can provide the character stream by cutting out the necessary fields from `shortlist` and piping the output to **paste**:

```
$ cut -d \| -f 1,4- shortlist | paste -d "|" cutlist1 -
a.k. shukla       |g.m.      |2233|sales      |12/12/52|6000
jai sharma        |director  |9876|production |12/03/50|7000
```

```
sumit chakrobarty|d.g.m.    |5678|marketing |19/04/43|6000
barun sengupta   |director  |2365|personnel |11/05/47|7800
n.k. gupta       |chairman  |5423|admin     |30/08/56|5400
```

You can also reverse the order of pasting by altering the location of the - sign:

```
cut -d "|" -f 1,4- shortlist | paste -d "|" - cutlist1
```

Joining Lines (-s) **paste** is more useful than you might think. Consider this address book that contains details of three persons, with three lines for each:

```
$ cat addressbook
anup kumar
anup_k@yahoo.com
24569083
vinod sharma
vinod_sharma@hotmail.com
34586532
madhuri_bahl
madhuri@heavens.com
39034943
```

Name
Email address
Telephone number

The **-s** option joins lines in the same way **vi**'s **J** command does *(7.5.4)*. Using this option on this file (with **paste -s addressbook**) would join all of these nine lines to form a single line. This won't be of much use, so we'll learn to use the **-d** option with multiple delimiters to join three lines at a time.

If we specify the delimiter string as **||\n** with **-d**, the delimiters are used in a circular manner. The first and second lines would be joined with the **|** as delimiter, and the same would be true for the second and third line. The third and fourth line would be separated by a newline. After the list is exhausted it is reused. This is exactly what we want:

```
$ paste -s -d"||\n" addressbook
anup kumar|anup_k@yahoo.com|24569083
vinod sharma|vinod_sharma@hotmail.com|34586532
madhuri_bahl|madhuri@heavens.com|39034943
```

Just see how **paste** works (with a single file this time) to concatenate lines in a specified manner. Table data is often split with each column on a separate line, and it's in situations like these that **paste** can be so useful.

12.7 sort: ORDERING A FILE

Sorting is the ordering of data in ascending or descending sequence. The **sort** command orders a file. Like **cut**, it identifies fields and it can sort on specified fields. We'll consider the important **sort** options by sorting the file **shortlist** (created in Section 12.5) in different ways. By default the entire line is sorted:

```
$ sort shortlist
2233|a.k. shukla      |g.m.     |sales     |12/12/52|6000
2365|barun sengupta   |director |personnel |11/05/47|7800
```

```
5423|n.k. gupta          |chairman |admin      |30/08/56|5400
5678|sumit chakrobarty|d.g.m.    |marketing  |19/04/43|6000
9876|jai sharma         |director |production|12/03/50|7000
```

By default, **sort** reorders lines in ASCII collating sequence—whitespace first, then numerals, uppercase letters and finally lowercase letters. This default sorting sequence can be altered by using certain options. You can also sort on one or more keys (fields) or use a different ordering rule.

12.7.1 sort Options

The important **sort** options are summarized in Table 12.1. We'll be using the -k (key) POSIX option to identify *keys* (the fields) instead of the *+n* and *-n* forms (where *n* is the field number) that were used earlier. Unlike **cut** and **paste**, **sort** uses one or more contiguous spaces as the default field separator (tab in **cut** and **paste**). We'll use the -t option to specify the delimiter.

Sorting on Primary Key (-k) Let's now use the -k option to sort on the second field (name). The option should be -k 2:

```
$ sort -t"|" -k 2 shortlist
2233|a.k. shukla         |g.m.      |sales      |12/12/52|6000
2365|barun sengupta      |director  |personnel  |11/05/47|7800
9876|jai sharma          |director  |production|12/03/50|7000
5423|n.k. gupta          |chairman  |admin      |30/08/56|5400
5678|sumit chakrobarty|d.g.m.    |marketing  |19/04/43|6000
```

The sort order can be reversed with the -r (reverse) option. The following sequence reverses a previous sorting order:

```
$ sort -t"|" -r -k 2 shortlist
5678|sumit chakrobarty|d.g.m.    |marketing  |19/04/43|6000
5423|n.k. gupta          |chairman  |admin      |30/08/56|5400
9876|jai sharma          |director  |production|12/03/50|7000
2365|barun sengupta      |director  |personnel  |11/05/47|7800
2233|a.k. shukla         |g.m.      |sales      |12/12/52|6000
```

sort combines options in a rather unusual way. The previous command sequence could also have been written as:

```
sort -t"|" -k 2r shortlist
```

Sorting on Secondary Key You can sort on more than one key, i.e., you can provide a secondary key to **sort**. If the primary key is the third field, and the secondary key is the second field, then you need to specify for every -k option, where the sort ends. This is done in this way:

```
$ sort -t"|" -k 3,3 -k 2,2 shortlist
5423|n.k. gupta          |chairman  |admin      |30/08/56|5400
5678|sumit chakrobarty|d.g.m.    |marketing  |19/04/43|6000
2365|barun sengupta      |director  |personnel  |11/05/47|7800
9876|jai sharma          |director  |production|12/03/50|7000
2233|a.k. shukla         |g.m.      |sales      |12/12/52|6000
```

This sorts the file by designation and name. `-k 3,3` indicates that sorting starts on the third field and ends on the same field.

Sorting on Columns You can also specify a character position within a field to be the beginning of sort. If you are to sort the file according to the year of birth, then you need to sort on the seventh and eighth column positions within the fifth field:

```
$ sort -t"|" -k 5.7,5.8 shortlist
5678|sumit chakrobarty|d.g.m.    |marketing |19/04/43|6000
2365|barun sengupta    |director |personnel |11/05/47|7800
9876|jai sharma        |director |production|12/03/50|7000
2233|a.k. shukla       |g.m.     |sales     |12/12/52|6000
5423|n.k. gupta        |chairman |admin     |30/08/56|5400
```

The `-k` option also uses the form `-k m.n` where *n* is the character position in the *m*th field. So `5.7,5.8` means that sorting starts on column 7 of the fifth field and ends on column 8.

Numeric Sort (-n) When **sort** acts on numerals, strange things can happen. When you sort a file containing only numbers, you get a curious result:

```
$ sort numfile
10
2
27
4
```

This is probably not what you expected, but the ASCII collating sequence places 1 above 2 and 2 above 4. That's why 10 preceded 2 and 27 preceded 4. This can be overridden by the `-n` (numeric) option:

```
$ sort -n numfile
2
4
10
27
```

Removing Repeated Lines (-u) The `-u` (unique) option lets you remove repeated lines from a file. If you "cut" out the designation field from `emp.lst`, you can pipe it to **sort** to find out the unique designations that occur in the file:

```
$ cut -d"|" -f3 emp.lst | sort -u | tee desigx.lst
chairman
d.g.m.
director
executive
g.m.
manager
```

We used three commands to solve a text manipulation problem. Here, **cut** select the third field from `shortlist` for **sort** to work on.

Other sort Options Even though **sort**'s output can be redirected to a file, we can use its **-o** option to specify the output filename. Curiously enough, the input and output filenames can even be the same:

```
sort -o sortedlist -k 3 shortlist      Output stored in sortedlist
sort -o shortlist shortlist            Output stored in same file
```

To check whether the file has actually been sorted in the default order, use the **-c** (check) option:

```
$ sort -c shortlist
$ _                                    File is sorted
```

You can also add the **-k** option to the above to check whether a specific field is sorted:

```
$ sort -t"|" -c -k 2 shortlist
sort: shortlist:2: disorder: 2365|barun sengupta   |director |personnel |11/05/4
7|7800
```

When **sort** is used with multiple filenames as arguments, it concatenates them and sorts them collectively. When large files are sorted in this way, performance often suffers. The **-m** (merge) option can merge two or more files that are sorted individually:

```
sort -m foo1 foo2 foo3
```

This command will run faster than the one used without the **-m** option only if the three files are sorted.

Tip: Commit to memory the default delimiter used by **cut**, **paste** and **sort**. **cut** and **paste** use the tab, but **sort** uses a contiguous string of spaces as a single delimiter.

Table 12.1 **sort** Options

Option	*Description*
-tchar	Uses delimiter *char* to identify fields
-kn	Sorts on *n*th field
-km,n	Starts sort on *m*th field and ends sort on *n*th field
-km.n	Starts sort on *n*th column of *m*th field
-u	Removes repeated lines
-n	Sorts numerically
-r	Reverses sort order
-f	Folds lowercase to equivalent uppercase (case-insensitive sort)
-m list	Merges sorted files in *list*
-c	Checks if file is sorted
-o flname	Places output in file *flname*

12.8 uniq: LOCATE REPEATED AND NONREPEATED LINES

When you concatenate or merge files, you'll face the problem of duplicate entries creeping in. We saw how **sort** removes them with the -u option. UNIX offers a special tool to handle these lines—the **uniq** command. Consider a sorted file dept.lst that includes repeated lines:

```
$ cat dept.lst
01|accounts|6213
01|accounts|6213
02|admin|5423
03|marketing|6521
03|marketing|6521
03|marketing|6521
04|personnel|2365
05|production|9876
06|sales|1006
```

uniq simply fetches one copy of each line and writes it to the standard output:

```
$ uniq dept.lst
01|accounts|6213
02|admin|5423
03|marketing|6521
04|personnel|2365
05|production|9876
06|sales|1006
```

Since **uniq** requires a sorted file as input, the general procedure is to sort a file and pipe the output to **uniq**. The following pipeline also produces the same output, except that the output is saved in a file:

```
sort dept.lst | uniq -  uniqlist
```

uniq is indeed unique; if provided with two filenames as arguments, **uniq** will read the first file and write its output to the second. Here, it reads from the standard input and writes to uniqlist.

12.8.1 uniq Options

To select unique lines, it's preferable to use **sort -u** that does the job with a single command. But **uniq** has a couple of useful options; they can be used to make simple database queries.

Selecting the Nonrepeated Lines (-u) To determine the designation that occurs uniquely in emp.lst, cut out the third field, sort it, and then pipe it to **uniq**. The -u (unique) option selects only those lines that are not repeated:

```
$ cut -d"|" -f3 emp.lst | sort | uniq -u
chairman
```

Selecting the Duplicate Lines (-d) The -d (duplicate) option selects only one copy of the repeated lines:

```
$ cut -d"|" -f3 emp.lst | sort | uniq -d
d.g.m.
director
executive
g.m.
manager
```

Counting Frequency of Occurrence (-c) The -c (count) option displays the frequency of occurrence of all lines, along with the lines:

```
$ cut -d"|" -f3 emp.lst | sort | uniq -c
   1 chairman
   2 d.g.m.
   4 director
   2 executive
   4 g.m.
   2 manager
```

In SQL, you would be using this:
SELECT COUNT(*), JOB FROM EMP
GROUP BY JOB ;

This is an extremely useful option, and we'll make best use of it in an example that is taken up at the end of this chapter. It raises the possibility of printing a word-count list that displays the frequency of occurrence of each word.

Caution: Like **sort**, **uniq** also accepts the output filename as an argument, but without using an option (unlike -o in **sort**). If you use **uniq foo1 foo2**, **uniq** simply processes foo1 and overwrites foo2 with its output. Never use **uniq** with two filenames unless you know what you are doing.

12.9 tr: TRANSLATING CHARACTERS

So far, the commands have been handling either entire lines, or columns. The **tr** (translate) filter manipulates *individual characters* in a line. More specifically, it translates characters using one or two compact expressions:

```
tr options expression1 expression2 standard input
```

Note that tr *takes input only from standard input; it doesn't take a filename as argument.* By default, it translates each character in *expression1* to its mapped counterpart in *expression2*. The first character in the first expression is replaced with the first character in the second expression, and similarly for the other characters.

Let's use **tr** to replace the | with a ~ (tilde) and the / with a -. Simply specify two expressions containing these characters in the proper sequence:

```
$ tr '|/' '~-' < emp.lst  | head -n 3
2233~a.k. shukla      ~g.m.      ~sales      ~12-12-52~6000
9876~jai sharma       ~director  ~production~12-03-50~7000
5678~sumit chakrobarty~d.g.m.    ~marketing  ~19-04-43~6000
```

Note that the lengths of the two expressions should be equal. If they are not, the longer expression will have unmapped characters (not in Linux). Single quotes are used here because no variable

evaluation or command substitution is involved. It's just as easy to define the two expressions as two separate variables, and then evaluate them in double quotes:

```
exp1='|/' ; exp2='--'
tr "$exp1" "$exp2" < emp.lst
```

Like wild-cards, **tr** also accepts ranges in the expressions. The same rules apply; the character on the right of the - must have an ASCII value higher than that of the character on the left. The escaping rules should also be obvious; the character [needs to be escaped if the special meaning is to be removed from it.

Changing Case of Text Since **tr** doesn't accept a filename as argument, the input has to be redirected from a file or a pipe. The following sequence changes the case of the first three lines from lower to upper:

```
$ head -n 3 emp.lst | tr '[a-z]' '[A-Z]'
2233|A.K. SHUKLA       |G.M.      |SALES      |12/12/52|6000
9876|JAI SHARMA        |DIRECTOR  |PRODUCTION|12/03/50|7000
5678|SUMIT CHAKROBARTY|D.G.M.    |MARKETING  |19/04/43|6000
```

Reversing the two expressions will convert case from upper to lower. **tr** is often used to change the case of a file's contents.

12.9.1 tr Options

Deleting Characters (-d) The file `emp.lst` has fields separated by delimiters and the date formatted in readable form with a /. In nondatabase setups, delimiters are not used, and the date is generally represented as a six-character field in the format *ddmmyy*. To convert this file to the traditional format, use the `-d` (delete) option to delete the characters | and / from the file. The following command does it for the first three lines:

```
$ tr -d '|/' < emp.lst | head -n 3
2233a.k. shukla      g.m.     sales     1212526000
9876jai sharma       director production1203507000
5678sumit chakrobartyd.g.m.   marketing 1904436000
```

Compressing Multiple Consecutive Characters (-s) UNIX tools work best with fields rather than columns (like above), so it's preferable to use files with delimited fields. In that case, lines need not be of fixed length; you can eliminate all redundant spaces with the `-s` (squeeze) option, which squeezes multiple consecutive occurrences of its argument to a single character. We can then have compressed output with lines in free format:

```
$ tr -s ' ' <emp.lst | head -n 3
2233|a.k. shukla |g.m. |sales |12/12/52|6000
9876|jai sharma |director |production|12/03/50|7000
5678|sumit chakrobarty|d.g.m. |marketing |19/04/43|6000
```

Tip: You can use the `-s` option to compress all contiguous spaces in the output of several UNIX commands and then use **cut** to extract individual fields from this compressed output. For instance, you can "cut" out any field from the listing.

Complementing Values of Expression (-c) Finally, the -c (complement) option complements the set of characters in the expression. Thus, to delete all characters except the | and /, you can combine the -d and -c options:

```
$ tr -cd '|/' <emp.lst
||||/||||||/||||||/||||||/||||||/||||||/||||||/||||||/||||||/||||||/||||||/||||
|//||||||/||||||/||||||/$ _
```

Unusual output indeed! **tr** has deleted all characters except the | and / from the file. The appearance of the prompt at the immediate end of output shows that the newline character has also not been spared. We'll use the -c and -d options to place each word in a separate line in our example section *(12.10)*.

Using ASCII Octal Values and Escape Sequences Like **echo**, **tr** also uses octal values and escape sequences to represent characters. This facility is specially suited for using nonprintable characters in the expression. So to have each field on a separate line, replace the | with the LF character (octal value 012):

```
$ tr '|' '\012' < emp.lst | head -n 6          Can also use \n
2233                                           instead of \012
a.k. shukla
g.m.
sales
12/12/52
6000
```

If you reverse the two expressions, you'll make the newline character visible. Study these **tr** options closely, and you'll discover many areas where you can apply them. We'll be using some of the **tr** options in the example that's considered next.

12.10 AN EXAMPLE: DISPLAYING A WORD-COUNT LIST

Armed with the knowledge of the basic UNIX filters, let's devise a command sequence that uses a number of filters in a pipeline. Document authors sometimes like to view a list that displays the frequency of usage of each word in the document. For this to be possible, each word has to be placed in a separate line. **tr** can do that by converting all spaces and tabs (octal 011) to newlines:

```
tr " \011" "\012\012" < fool                   Space is \040
```

There's a space before \011; we won't be displaying the symbol subsequently. If we define a word as a contiguous group of alphabetic characters, we have to use **tr** again to delete all nonalphabetic characters (apart from the newline) from the output of the first **tr** command. This requires the use of the complementary (-c) and delete (-d) options:

```
tr " \011" "\012\012" < fool | tr -cd "[a-zA-Z\012]"
```

You now have a list of words, with each word on a separate line. Now sort this output and pipe it to **uniq -c**:

```
$ tr " \011" "\012\012" < fool | tr -cd "[a-zA-Z\012]" | sort | uniq -c
     32 Apache                                    Apache used 32 times
     18 DNS
     10 Directory
     16 FQDN
     25 addresses
     56 directory
```

You had to use four commands to display the word count. You'll need two more to sort the list in reverse numeric sequence and print it in three columns:

```
$ tr " \011" "\012\012" < fool | tr -cd "[a-zA-Z\012]" | sort | uniq -c \
> | sort -nr | pr -t -3
     56 directory          25 addresses          16 FQDN
     32 Apache             18 DNS                10 Directory
```

For the sake of readability, we split the command line into two lines by using \ to escape the *[Enter]* key.

12.11 CONCLUSION

This chapter presented some of the commonly used filters available in the UNIX system. These filters mostly work on entire lines or fields; it's only **tr** that manipulates individual characters. These filters have limited use when they are used in standalone mode. But you also used a number of them in pipelines to perform tasks that apparently seem so difficult to achieve by conventional means. We still have four other filters to discuss—**grep**, **sed**, **awk** and **perl**, but three of them actually deviate from the do-one-thing-well approach to UNIX tool building.

WRAP UP

The **pr** command formats input to print headings and page numbers but can also drop them (-t). The output can be numbered (-n), doublespaced (-d) and offset from the left (-o).

head displays the beginning of a file, while **tail** displays the end. Unlike **head**, **tail** can also be used with a line number (with the + option) from where extraction should begin. It is most useful in monitoring the growth of a file (-f).

cut selects columns (-c) from its input, as well as fields (-f). You can join two files laterally with **paste**. By using the delimiter in a circular manner, **paste** can join multiple lines into one.

Using **sort**, you can sort on one or more fields or keys (-k), and columns within these fields. You can sort numerically (-n), reverse the sort order (-r), make a case-insensitive sort (-f), and remove repeated lines (-u).

uniq removes repeated or nonrepeated lines. The command is often combined with **sort** to order the input first.

tr translates characters using two expressions, but accepts *only* standard input. It can be used to change the case of letters. You can compress multiple consecutive occurrences (-s) or delete a specific character (-d). You can also use it with ASCII octal values and escape sequences to transform nonprintable characters.